

# NAVAL POSTGRADUATE SCHOOL

## Monterey, California



### THESIS

#### SUPPORTING A TRUSTED PATH FOR THE LINUX OPERATING SYSTEM

by

Scott A. Bartram

June 2000

Thesis Advisor:  
Co-Advisor:

Cynthia E. Irvine  
Paul C. Clark

Approved for public release; distribution is unlimited.

DMC QUALITY INSPECTED 4

20000818 067

# REPORT DOCUMENTATION PAGE

Form Approved  
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

<b>1. AGENCY USE ONLY (Leave blank)</b>		<b>2. REPORT DATE</b> June 2000	<b>3. REPORT TYPE AND DATES COVERED</b> Master's Thesis
<b>4. TITLE AND SUBTITLE</b> Supporting A Trusted Path For The Linux Operating System			<b>5. FUNDING NUMBERS</b>
<b>6. AUTHOR(S)</b> Bartram, Scott A.			
<b>7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)</b> Naval Postgraduate School Monterey, CA 93943-5000			<b>8. PERFORMING ORGANIZATION REPORT NUMBER</b>
<b>9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)</b>			<b>10. SPONSORING / MONITORING AGENCY REPORT NUMBER</b>
<b>11. SUPPLEMENTARY NOTES</b> The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
<b>12a. DISTRIBUTION / AVAILABILITY STATEMENT</b> Approved for public release; distribution is unlimited.			<b>12b. DISTRIBUTION CODE</b>
<b>13. ABSTRACT (maximum 200 words)</b> The existence of Trojan horses, viruses, and other malicious software has motivated the computer security industry to invent mechanisms that protect against malicious software. One such mechanism is called the Trusted Path. The Trusted Path provides a way for the system to authenticate itself to the user. Once invoked, the Trusted Path provides an environment in which the user can perform trusted operations such as login, logout, and change password. This thesis provides a high level design for a Trusted Path and an in depth analysis of how a Trusted Path can be implemented in the Linux operating system. Research of process family creation and keyboard handling has led to the implementation of a Secure Attention Key that can be used to invoke a Trusted Path in Linux. This research is meant to be used in combination with other efforts to enhance the Linux operating system as an inexpensive platform for instruction on computer security policies.			
<b>14. SUBJECT TERMS</b> Trusted Path, Secure Attention Key, Computer Security, Linux, Policy Enhanced Linux			<b>15. NUMBER OF PAGES</b> 80
			<b>16. PRICE CODE</b>
<b>17. SECURITY CLASSIFICATION OF REPORT</b> Unclassified	<b>18. SECURITY CLASSIFICATION OF THIS PAGE</b> Unclassified	<b>19. SECURITY CLASSIFICATION OF ABSTRACT</b> Unclassified	<b>20. LIMITATION OF ABSTRACT</b> UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)  
Prescribed by ANSI Std. Z39-18



Approved for public release; distribution is unlimited.

**SUPPORTING A TRUSTED PATH FOR THE LINUX OPERATING SYSTEM**

Scott A. Bartram  
Ensign, United States Navy  
B.S., Oregon State University, 1999

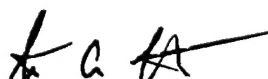
Submitted in partial fulfillment of the  
requirements for the degree of

**MASTER OF SCIENCE IN COMPUTER SCIENCE**

from the

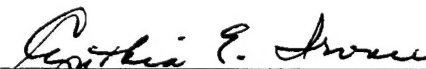
**NAVAL POSTGRADUATE SCHOOL  
June 2000**

Author:

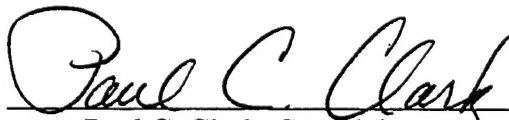


Scott A. Bartram

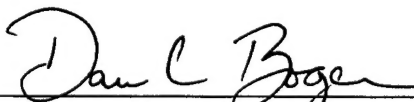
Approved by:



Cynthia E. Irvine, Thesis Advisor



Paul C. Clark, Co-Advisor



Dan Boger, Chairman  
Department of Computer Science



## **ABSTRACT**

The existence of Trojan horses, viruses, and other malicious software has motivated the computer security industry to invent mechanisms that protect against malicious software. One such mechanism is called the Trusted Path. The Trusted Path provides a way for the system to authenticate itself to the user. Once invoked, the Trusted Path provides an environment in which the user can perform trusted operations such as login, logout, and change password.

This thesis provides a high level design for a Trusted Path and an in depth analysis of how a Trusted Path can be implemented in the Linux operating system. Research of process family creation and keyboard handling has led to the implementation of a Secure Attention Key that can be used to invoke a Trusted Path in Linux.

This research is meant to be used in combination with other efforts to enhance the Linux operating system as an inexpensive platform for instruction on computer security policies.



## TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	COMPUTER SECURITY.....	1
B.	SECURE COMPUTER SYSTEM EVALUATION CRITERIA.....	2
C.	IDENTIFICATION AND AUTHENTICATION.....	4
D.	THE TRUSTED PATH.....	6
1.	Microsoft Windows NT .....	7
2.	Trusted Solaris.....	9
3.	XTS-300 .....	11
E.	POLICY ENHANCED LINUX .....	14
II.	TRUSTED PATH HIGH LEVEL DESIGN.....	15
A.	USER INTERFACE.....	15
1.	Start .....	17
2.	Ready.....	17
3.	Login Prompt.....	17
4.	Password Prompt.....	17
5.	Trusted Path Menu .....	18
6.	Help .....	19
7.	Session Level Prompt.....	19
8.	Change Password .....	19
9.	New Password .....	20
10.	Confirm and Change .....	20
11.	Run .....	20
12.	List Sessions .....	20
13.	Kill Session Menu .....	21
III.	TRUSTED PATH HIGH LEVEL DESIGN FOR LINUX.....	23
A.	USER INTERFACE.....	23
1.	Red Hat Linux 6.0 Getty/Login.....	23
2.	Modifications Needed .....	28
B.	SECURE ATTENTION KEY (SAK) .....	34
1.	Red Hat Linux 6.0 SAK .....	34
2.	Modifications Needed .....	41
IV.	IMPLEMENTATION STATUS AND FUTURE WORK.....	43
A.	IMPLEMENTATION CONSIDERATIONS .....	43
B.	PROBLEMS ENCOUNTERED .....	45
C.	FUTURE RESEARCH .....	45
1.	Login_Driver .....	45
2.	Session_Driver .....	46



3. Trusted_Path_Menu_Driver .....	46
4. Inittab.....	46
5. Administrative Role .....	46
APPENDIX A. MODULE DESIGN .....	49
A. STATE_MODULE .....	49
1. SetTPState .....	50
2. GetTPState .....	51
APPENDIX B. SOURCE CODE .....	53
A. SAK_DRIVER .....	53
1. Sak.h.....	53
2. Sak.c .....	53
B. MODIFIED KEYBOARD DRIVER FILES .....	55
1. Keyboard.c.....	55
2. Pc_keyb.c.....	56
C. MODIFIED CONFIGURATION SCRIPT .....	56
1. Kconfig.tk.....	56
D. DUMBGETTY .....	57
1. Dumbgetty.c .....	57
E. MODIFIED INITTAB .....	59
1. Inittab.....	59
LIST OF REFERENCES .....	61
INITIAL DISTRIBUTION LIST .....	65

## **I. INTRODUCTION**

### **A. COMPUTER SECURITY**

The need for computer security is growing at an astounding rate, parallel with the Internet. The speed and ease of transferring data from one site to another is increasing, yet Internet users are realizing that the World Wide Web was not designed to protect data being transferred across the network. Electrical engineers are designing new communication media with high bandwidth, low noise, and low attenuation (e.g., fiber optics). Network engineers are designing better methods of relaying data between network hubs, bridges, and routers. Software engineers are designing better network and application protocols for using the data. Consequently, massive amounts of data are being transferred with virtually no protection. It is easy to transfer program files from a random Internet site and run them on a local computer. There are some web pages that automatically transfer and execute files on an accessing computer without the user's knowledge. Even with an up to date virus checker, a user cannot be confident that there is not a virus or a Trojan horse in a recently downloaded file. The existence of viruses, Trojan horses, and other types of malicious software is the motivation for this thesis. Computer security is very important to the future of computing, and standards have been set to ensure the survival of computer security in the new millennium.

A computer with an operating system that enforces a security policy must be designed using the standards described in the government evaluation criteria described in the following section. This thesis discusses some of the essential mechanisms that are

needed in the design of the operating system for a secure computer. Particular emphasis is placed on accountability mechanisms such as Identification and Authentication and Trusted Path.

## **B. SECURE COMPUTER SYSTEM EVALUATION CRITERIA**

An early notable effort in computer security standards came in 1983 with the release of the Department of Defense Trusted Computer System Evaluation Criteria (TCSEC), commonly referred to as the "Orange Book" [Ref.1]. The most recent version of the Orange Book was released in 1985 and it lists the following purpose:

...\* To provide a standard to manufacturers as to what security features to build into their new and planned, commercial products in order to provide widely available systems that satisfy trust requirements (with particular emphasis on preventing the disclosure of data) for sensitive applications.  
\* To provide DoD Components with a metric with which to evaluate the degree of trust that can be placed in computer systems for the secure processing of classified and other sensitive information. \* To provide a basis for specifying security requirements in acquisition specifications.  
[Ref.1, p.2]

The method used for evaluating a computer system against TCSEC was, and for some ongoing evaluations still is, the Trusted Product Evaluation Program (TPEP) [Ref.2]. TPEP evaluates systems and then rates them with one of seven TCSEC evaluation classes. The TCSEC classes are labeled from D1 to A1, with D1 being the lowest (least secure) class. Each TCSEC class contains specific requirements for security policy, accountability, assurance, and documentation [Ref.1, p.5]. TPEP was sponsored by the National Computer Security Center (NCSC) but is now being replaced with the

Trust Technology Assessment Program (TTAP) [Ref.3]. TTAP is a program sponsored by the National Security Agency (NSA) that provides a means for commercial-off-the-shelf (COTS) products to be evaluated. TTAP oversees the establishment and regulation of TTAP Evaluation Facilities (TEF) [Ref.4]. Laboratories that want to become TEFs go through an application and evaluation process in order to be certified by the TTAP [Ref.3]. In addition to being a new program, TTAP also uses new evaluation criteria called the Common Criteria for Information Technology Security Evaluation (CCITSE or CC) [Ref.5].

The CC differs from TCSEC in that it does not group all of the security requirements (policy, accountability, and assurance) into one class. The CC permits separate evaluation of Protection Profiles (PP) [Ref.6] (a conceptual design including policy and accountability) or Security Targets (ST) [Ref.7] (an implemented system design including policy and accountability). After the evaluation, the PP or ST is assigned an Evaluation Assurance Level (EAL) [Ref.8] that is used as a marker for the level of assurance that should be placed in the PP or ST. It is easy to be misled and compare EALs (assurance only) directly to TCSEC classes that contain policy, accountability, and assurance. It is the PP or ST that contains the security policy and accountability requirements previously included in the TCSEC classes. Therefore, even though a PP may have a high EAL, it may compare to class D1 in TCSEC. It is important to be sure the PP fits the security policy that is needed by the system. One of the current efforts of the TTAP is the evaluation of PPs that contain the requirements from popular TCSEC classes. The difficulty when making these new PPs is that the CC contains a lot

more detail in its evaluation criteria and some of the TCSEC requirements are somewhat vague in comparison.

### **C. IDENTIFICATION AND AUTHENTICATION**

Computers that provide access to users commonly have a mechanism that provides assurance to the system that the user is not an impostor (i.e., username and password entry). This mechanism is called "Identification and Authentication". The TCSEC section on Identification and Authentication states:

The TCB [Trusted Computing Base] shall require users to identify themselves to it before beginning to perform any other actions that the TCB is expected to mediate. Furthermore, the TCB shall use a protected mechanism (e.g., passwords) to authenticate the user's identity. The TCB shall protect authentication data so that it cannot be accessed by any unauthorized user. [Ref.1, p.12]

Computer systems to be evaluated by the TCSEC are required to provide Identification and Authentication at classes of C1 and higher. Class C1 is defined as providing low protection discretionary access control. The CC section on Identification and Authentication states:

Families in this class [Class FIA: Identification and Authentication] address the requirements for functions to establish and verify a claimed user identity. Identification and Authentication is required to ensure that users are associated with the proper security attributes (e.g. identity, groups, roles, security or integrity levels). The unambiguous identification of authorized users and the correct association of security attributes with users and subjects is critical to the enforcement of the intended security policies. [Ref.5, Part 2, p.79]

The entry of a username and password to gain access to a computer is a familiar form of Identification and Authentication. The problem with many Identification and Authentication implementations is that they do not provide a mechanism for the system to authenticate itself to the user. It is very easy for an attacker to write a program that can look and feel exactly like the login process for a given computer. If it is easy to look and feel like a login program, then it is just as easy to steal a password, store it somewhere for later retrieval, print an error message to the console, and then start the “real” login program. The user would think the error message was caused by a typo and never suspect that the password was stolen. Figure 1 is an example of a login terminal for the Linux operating system.

```
Red Hat Linux release 6.0 (Hedwig)
Kernel 2.2.5-15 on an i686

login: bartram
Password:
[bartram@rudi bartram]$
```

Figure 1. Example Login Terminal

The scenario described above is called “login spoofing” which is accomplished using “masquerading software”. Masquerading software can be used when an attacker wants to obtain information by posing as a familiar process. The existence of masquerading software is one of the motivations for the design of the Trusted Path.

#### D. THE TRUSTED PATH

So that the user has a high level of assurance that the system being accessed is the “real” system rather than masquerading software, the system must have a Trusted Path. The Trusted Path is a mechanism that provides a way for the system to authenticate itself to the user. The TCSEC requires the Trusted Path at Class B2 and higher [Ref.1, p.29]. The TCSEC definition of the Trusted Path is, “A mechanism by which a person at a terminal can communicate directly with the Trusted Computing Base. This mechanism can only be activated by the person or the Trusted Computing Base and cannot be imitated by untrusted software” [Ref.1, p.117]. The CC section on the Trusted Path states:

*A **trusted path** provides a means for users to perform functions through an assured direct interaction with the TSF [Target of Evaluation Security Functions]. Trusted path is usually desired for user actions such as initial identification and/or authentication, but may also be desired at other times during a user’s session. Trusted path exchanges may be initiated by a user or the TSF. User responses via the trusted path are guaranteed to be protected from modification by or disclosure to untrusted applications. [Ref.5, Part2, p.167].*

The mechanism that is normally used to invoke a Trusted Path is called the Secure Attention Key (SAK) [Ref.9, p.67]. Traditionally, the Secure Attention Key has been implemented using a sequence of keystrokes from a keyboard connected to a system terminal. The TCSEC and CC do not reference the term “Secure Attention Key”; they only suggest that there must be a way for the user to initiate a Trusted Path. Turning the computer OFF and then ON again could be considered crude form of SAK. When the

Trusted Path is invoked (i.e., using the SAK), all other user processes using the terminal are suspended or killed (depending on the implementation or state of the system) while the Trusted Path remains active. Some Trusted Paths give the user options such as logout, shutdown, restart, and end task (kill process). The options that are provided to the user may support some other reasons for having a Trusted Path such as the ability to escape from an unwanted state (i.e., endless loop, deadlock), change a password, change session level, and modify access privileges. The following sections describe a few evaluated systems that have implemented a Trusted Path.

#### **1. Microsoft Windows NT**

Microsoft Windows NT Workstation and Windows NT Server Version 4.0 with Service Pack 6a and C2 Update has been rated Class C2 by the National Security Agency (NSA) in accordance with the TCSEC [Ref.10]. The rating was announced in November 1999 and was conducted by the Science Applications International Corporation (SAIC), Center for Information Security Technology, Evaluation Laboratory. The version of Windows NT mentioned above is the most recently evaluated version and has evolved from the previous version 3.5 that was rated Class C2 with Service Pack 3 [Ref.11]. The evaluation summary for version 3.5 included the following remarks regarding Trusted Path:



... the Windows NT platform was examined against the B2 Trusted Path and B2 Trusted Facility Management functional requirements of the TCSEC. A system that satisfies the B2 Trusted Path functional requirement supports a trusted communication path between itself and the user for identification and authentication. A system that satisfies the B2 Trusted Facility Management functional requirements supports the ability to separate operator and administrator functions. Although the Windows NT platform satisfies these functional requirements at the B2 level, it was not evaluated against any assurance requirements above its rated C2 level. [Ref.11]

Windows NT uses a window to notify the user that the SAK must be pressed in order to start a user session. The window displays: "Press Ctrl+Alt+Delete to log on". The SAK for Windows NT is the simultaneous key sequence "Control", "Alt", and "Delete". When the SAK is pressed the user is presented a Logon Information (Trusted Path) window that prompts for a username and password. The Logon Information window has the following options:

- OK
- Cancel
- Help
- Shut Down

After the user has successfully logged onto the system, the Trusted Path offers different options to the user. When the SAK is pressed, the Windows NT Security (Trusted Path) window is presented with current logon information and the following options:

- Lock Workstation
- Logoff

- Shut Down
- Change Password
- Task Manager
- Cancel

By pressing the SAK, the user is given assurance that they are actually communicating with the system and not to masquerading software. Important tasks such as password entry and password changing are only available to regular users after invoking the Trusted Path.

## **2. Trusted Solaris**

Trusted Solaris 7 is the most recent version of the system offered by Sun Microsystems, Inc. and is entering into evaluation under the Common Criteria with the goal of receiving an EAL4 evaluation class for Trusted Solaris 8 shortly after the product is released. Trusted Solaris 7 will not receive an evaluation class [Ref.12]. The Trusted Solaris 7 is derived from previous versions of Trusted Solaris that have been evaluated by the National Computer Security Center (NCSC), the Defense Intelligence Agency (DIA), and the United Kingdom Information Technology Security (UK ITSEC) scheme. The Trusted Solaris system was originally designed to be evaluated as a compartmented mode workstation (CMW) system. A system evaluated as meeting DIA criteria for a CMW system also has a Class B1 rating according to the NCSC criteria (TCSEC) [Ref.13, p.4]. As is with Windows NT, even though Trusted Solaris has not been evaluated as a Class B2 system, it has implemented a Trusted Path.

The Trusted Solaris Trusted Path is different in that it does not use the keyboard for its SAK. All of the Trusted Path options are accessed via a windows-based Graphical User Interface (GUI) and a mouse (the mouse is used as the SAK). The Trusted Path is activated in one of two ways: from the label stripe of a window or from the screenstripe [Ref.14, p.10]. The screenstripe (always visible) is located at the bottom of the screen and displays information based on what is happening with the mouse pointer, the keyboard, and the windows on the screen. This section focuses only on the Trusted Path for the Trusted Solaris. The “Security Features User’s Guide to Trusted Solaris” provides complete explanations of security features available via the Trusted Path [Ref.13]. When the MENU button on the mouse is pressed while hovering over the label stripe of a window, the following Trusted Path options are presented:

- Show Full Window Label – shows the information and sensitivity labels of the window [Ref.14, p.60].
- Set Input Information Label – used to set the input information label from the keyboard [Ref.14, p.61].

When the MENU button on the mouse is pressed while hovering over the screenstripe, the Trusted Path Menu appears.

The Trusted Path Menu is divided into two main sections: Utilities and Set Labels. The initial Trusted Path Menu window that is displayed contains the Utilities and Set Labels directories along with options for Change Password and Set Screen Access. Table 1 lists and describes the Trusted Path Menu options for the Utilities and Set Labels directories [Ref.14, p.62-68].

<b>Utilities</b>	<b>Description</b>
Refresh Screen	Redraw the screen
Windows Controls	Open/close, full/restore size, quit
Save Workspace	Save current window configuration
Lock Screen	Password protect workspace
Log Out	Terminate session
<b>Set Labels</b>	<b>Description</b>
File and Directories	Set file labels
Workspace Menu SL	Set workspace menu label
Cut and Paste Defaults	Display selection labeler

Table 1. Trusted Solaris Trusted Path Menu Options

A major difference between the Trusted Solaris and the other two systems described in this chapter is the login process. The user is not given an option to enter the Trusted Path before entering login information. The Trusted Solaris Security Features User's Guide states:

When the login tool or lockscreen application is running, the screenstripe is not visible; instead, the entire background, including the area of the screenstripe, displays a trusted pattern. [Ref.14, p.10]

The guide does not describe what the trusted pattern is supposed to look like. However, since only the Trusted part of the system has the authority to cover up the screenstripe (which the login tool does), the user can feel confident that the system is in a Trusted state.

### 3. XTS-300

The XTS-300 is a combination of STOP, a multilevel secure operating system, and an Intel x86 hardware base (supplied by Wang Government Services, Inc.). XTS-300 STOP 4.4.2 has been rated Class B3 by the NSA in accordance with the TCSEC [Ref.15].

The XTS-300 STOP 4.4.2 is the most recently evaluated version of the product (March 1998) and has evolved from the XTS-200 STOP 3.1.E that was rated Class B3 in May 1992 [Ref.16]. The XTS-200 evolved from the SCOMP Version STOP 2.1 that was rated Class A1 in December 1984 [Ref.16]. Since the XTS-300 has been evaluated at Class B3 and the Trusted Path is required at Classes greater than or equal to B2, it follows that there is a Trusted Path on the XTS-300.

Unlike Windows NT, the XTS-300 does not provide notification to the user that the SAK must be pressed in order to start a session. After the system is booted, there is no way for the user to communicate with the system until the SAK has been pressed. The SAK for the XTS-300 console is the simultaneous key sequence "Alt" and "SysRq" [Ref.17, p.9]. When the SAK is pressed, the user is connected with the Secure Server. The Secure Server is that portion of Trusted Software that processes terminal commands [Ref.17, p.9]. When first logging into the system, the Secure Server prompts the user to enter a username and password [Ref.17, p.11]. After the user has successfully logged onto the system, the Secure Server (Trusted Path) offers different options to the user. When the SAK is pressed, the user is connected with the Secure Server in the trusted environment and prompted to enter a command [Ref.17, p.14]. The commands available to the user are called TCB (Trusted Computing Base) User Commands and they are split into 4 categories: Process Management, Session Management, File Management, and System Status. Table 2 lists and describes the TCB User Commands for each category [Ref.17, p.15-38].

<b>Process Management</b>	<b>Description</b>
Disconnect	Disassociate a process family from a user session (active processes will continue to run after logout)
Ikill	Immediately kill all processes in specified family (guarantees termination of the process family)
Kill	Kill all processes in specified family (does not guarantee termination of the process family)
Logout	Log off system
Reattach	Reconnect to a process family (return to an existing UNIX session)
Run	Begin execution of a process family (leave trusted environment and start a UNIX shell)
<b>Session Management</b>	<b>Description (used for setting up UNIX shell)</b>
Ccp	Change command processor pathname
Cdl	Change default level
Chd	Change home directory pathname
Cup	Change user password
Sg	Set group identifier for the current session
Sl	Set access level for the current session
<b>File Management</b>	<b>Description</b>
Fsm	Provides various functions for displaying, modifying, or deleting file system objects
<b>System Status</b>	<b>Description</b>
Session	Display information about the current session
System	Display system status information

Table 2. XTS-300 TCB User Commands Available Through The Trusted Path

Table 2 reflects only the commands available to the user. There are 3 distinct roles in the XTS-300 environment:

- System administrator
- Operator
- User

It is important to note that there are several other TCB commands that are available to people with higher privilege levels (system administrators and operators). Trusted activities such as configuring, bypassing, activating, or deactivating security features are available to administrators and operators only[Ref.18, p.1].

The XTS-300 supports an X-Windows (GUI) environment for use outside of the TCB. When the SAK is pressed during an X-Windows session, the X-Windows process family is detached and the Secure Server prompt appears. The Trusted Environment for the XTS-300, though not as GUI oriented as Windows NT, provides a much larger variety of options to the user.

Also, the XTS-300 contains functionality similar to the Linux operating system (command shell support, X-windows, and a UNIX compatible non-trusted environment) which makes it a good reference for the design of a Trusted Path for Linux.

#### **E. POLICY ENHANCED LINUX**

Clark describes a low cost approach to implementing a Mandatory Access Control (MAC) security policy using the Linux operating system [Ref.19]. In his conclusions, Clark suggests the implementation of a Trusted Path for use with the version of Policy Enhanced Linux resulting from his work. The remainder of this thesis is dedicated to providing the design and implementation of a Trusted Path for Policy Enhanced Linux [Ref.19]

## **II. TRUSTED PATH HIGH LEVEL DESIGN**

This chapter provides a high level description of the trusted path user interface. Chapter III describes lower level Linux modifications to implement the design.

### **A. USER INTERFACE**

There are a number of different states from which the user can apply the Secure Attention Key to invoke the Trusted Path. Consequently, invocation of the Trusted Path can lead to more than one state within the Trusted Path itself. More specifically, the SAK does not always bring the user to the same interface, rather it is dependent upon the state the system is in at the time the SAK is pressed. It is important to account for each possible system state and then which interface should be used. Figure 2 displays the Finite State Machine for the User Interface.



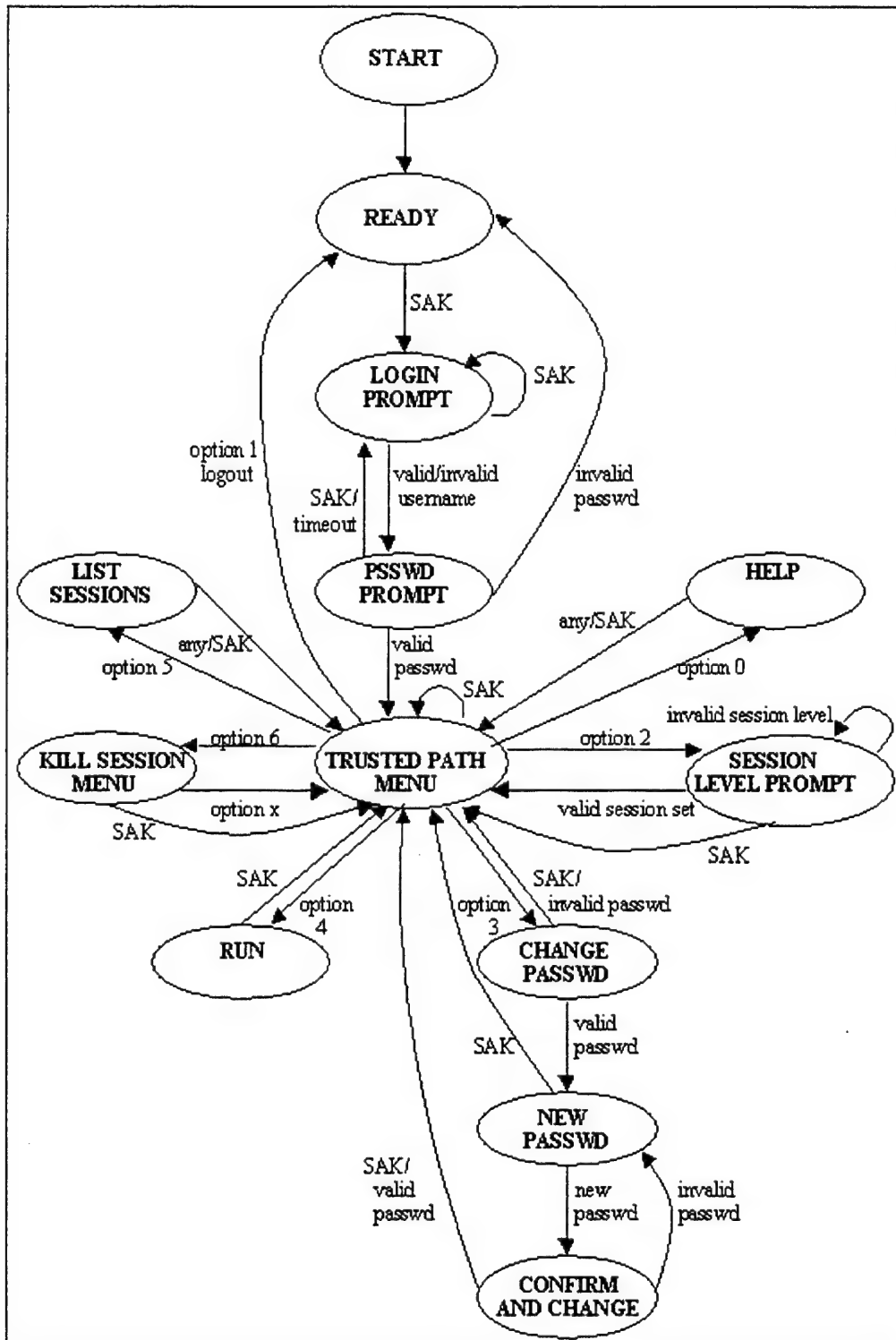


Figure 2. User Interface Finite State Machine

### **1. Start**

This state covers the entire boot and initialization process of the system. If the SAK is pressed while the system is in the Start state, nothing will happen and the system will continue initializing.

### **2. Ready**

Once the operating system finishes its initialization and all of the appropriate processes are running, the system enters the Ready state in which the user is given a prompt. At this point the user must use the Trusted Path in order to gain access to the system. This state will display a message to the user that asks for the SAK to be pressed in order to start. Only the SAK will allow the system to start the Trusted Path and start the login process.

### **3. Login Prompt**

This state is reached from the Ready state or from the Password Prompt state after the SAK has been pressed. There will be a display to the user that shows the trusted path is active. Also, the login prompt will be displayed requesting a username. At this point if the SAK is pressed, the Trusted Path will restart and the previously mentioned displays will be redisplayed. When a username (valid or invalid) is entered, the Trusted Path moves to the Password Prompt state.

### **4. Password Prompt**

This state is reached from the Login Prompt after a username has been entered. There will be a display to the user that shows the trusted path is active. Also, the password prompt will be displayed requesting a password. At this point if the SAK is

pressed, the Trusted Path will move to the Login Prompt state. If a valid password is entered, the Trusted Path will move to the Trusted Path Menu. If a password has not been entered in a reasonable amount of time (the timeout interval), the user is returned to the Login Prompt. If an invalid password is entered, the user is returned to the Ready state and is prompted to press the SAK again.

## **5. Trusted Path Menu**

This state is reached from many states and is the workhorse of the Trusted Path. There will be a display to the user that shows the trusted path is active and the current session level being used. Also, there will be a prompt for the user to select from the following menu options:

- 0 – Help
- 1 – Logout
- 2 – Session Level
- 3 – Change Password
- 4 – Run
- 5 – List Sessions
- 6 – Kill Session

Other options such as reboot and halt could be added as administrator option functions presented to those who are configured as administrators. At this point if the SAK is pressed, the Trusted Path Menu will be redisplayed. Selection of a particular option will move the Trusted Path to the associated state.

## **6. Help**

This state is reached from the Trusted Path Menu (Option 0). There will be a display to the user that shows the trusted path is active. The user will be provided with text explaining how the Trusted Path Menu should be used. At this point if the SAK or any other key is pressed, the user is returned to the Trusted Path Menu.

## **7. Session Level Prompt**

This state is reached from the Trusted Path Menu (Option 2). There will be a display to the user that shows the trusted path is active. There will be a user prompt and an example of how to specify a session level (i.e., how to type in the level and any categories). At this point if the SAK is pressed, the user is returned to the Trusted Path Menu without changing the session level. When an invalid session level is entered or the level is outside the user's clearance, the current session level is not changed, a message is displayed telling the user what was done incorrectly, and the user is returned to the Session Level Menu. When a valid session level is entered, the current session level is set to the entered session level and the user is returned to the Trusted Path Menu.

## **8. Change Password**

This state is reached from the Trusted Path Menu (Option 3). There will be a display to the user that shows the trusted path is active. The user will be prompted to enter the old password. At this point if the SAK is pressed or an invalid password is entered, the user is returned to the Trusted Path Menu without changing the password. After the old password is entered and verified to be true, the Trusted Path moves to the New Password state.

## **9. New Password**

This state is reached by typing a valid password at the Change Password state. There will be a display to the user that shows the trusted path is active. The user will be prompted to enter a new password. At this point if the SAK is pressed, the user is returned to the Trusted Path Menu without changing the password. After the new password is entered, the Trusted Path moves to the Confirm and Change state.

## **10. Confirm and Change**

This state is reached by typing a new password at the New Password state. There will be a display to the user that shows the trusted path is active. The user will be prompted to re-enter the new password. At this point if the SAK is pressed, the user is returned to the Trusted Path Menu without changing the password. If the password entered is valid, the password is changed and the user is returned to the Trusted Path Menu. If the password entered is invalid, the user is returned to the New Password State.

## **11. Run**

This state is reached from the Trusted Path Menu (Option 4). A user shell will be created at the current session level and the user will no longer be informed that the trusted path is active. The SAK is the only way to get back to the Trusted Path Menu from this state.

## **12. List Sessions**

This state is reached from the Trusted Path Menu (Option 5). There will be a display to the user that shows the trusted path is active. The user will be provided with a list of sessions that are currently available. The list of sessions will display the name of

the sessions and the associated levels. At this point if the SAK or any other key is pressed, the user will be returned to the Trusted Path Menu.

### **13. Kill Session Menu**

This state is reached from the Trusted Path Menu (Option 6). There will be a display to the user that shows the trusted path is active. There will be a prompt for the user to select from the following menu options:

0 – go back

1 – kill session level 1

2 – kill session level 2

... and so on depending on the number of open session levels

At this point if the SAK is pressed, the user will be taken back to the Trusted Path Menu and no sessions will be killed. Selection of a particular option will kill the associated session.

THIS PAGE INTENTIONALLY LEFT BLANK

### **III. TRUSTED PATH HIGH LEVEL DESIGN FOR LINUX**

This chapter describes the current design (login, SAK, process management) used by Red Hat Linux 6.0 and the modifications needed to implement the high level design from the previous chapter.

#### **A. USER INTERFACE**

##### **1. Red Hat Linux 6.0 Getty/Login**

Linux currently uses the user applications 'mingetty' and 'login' to start a command shell. The 'mingetty' application is spawned by the first process in the Linux Process Table, called 'init' [Ref.20, p.389]. The 'init' process is assigned process identification number (PID) 1. All processes that are created and placed on the Linux Process Table are children of 'init'. The 'mingetty' application is started after 'init' starts several other processes that are required for the operating system to function properly. This section only discusses the processes necessary for a user to login to the system.

'Init' and other (parent) processes are able to create new (child) processes by using the 'fork' system call [Ref.20, p.306]. The 'fork' system call generates a child process as a copy of the parent process. When the copy is made, a new PID is assigned to the child process and only the page tables and process structure are duplicated. Once a child process has been created, the 'execve' system call can be used to execute a new program using the task structure of the child [Ref.20, p.345]. More specifically, 'execve' overlays the calling (child) process by overwriting the calling process' code segment with the loaded program. The program inherits the PID of the calling process and any pending



signals are deleted. Basically, the calling process is replaced by the loaded program to create a new process that recycles the PID and task structure of the calling process. The system calls 'fork' and 'execve' are used to add to the Linux Process Table and as such are instrumental in the following description of the user login process. Figure 3 displays the User Login Finite State Machine and the associated Process Flow Diagram. The dotted lines represent the areas where the Process Flow Diagram and User Login FSM are directly related (correlation).

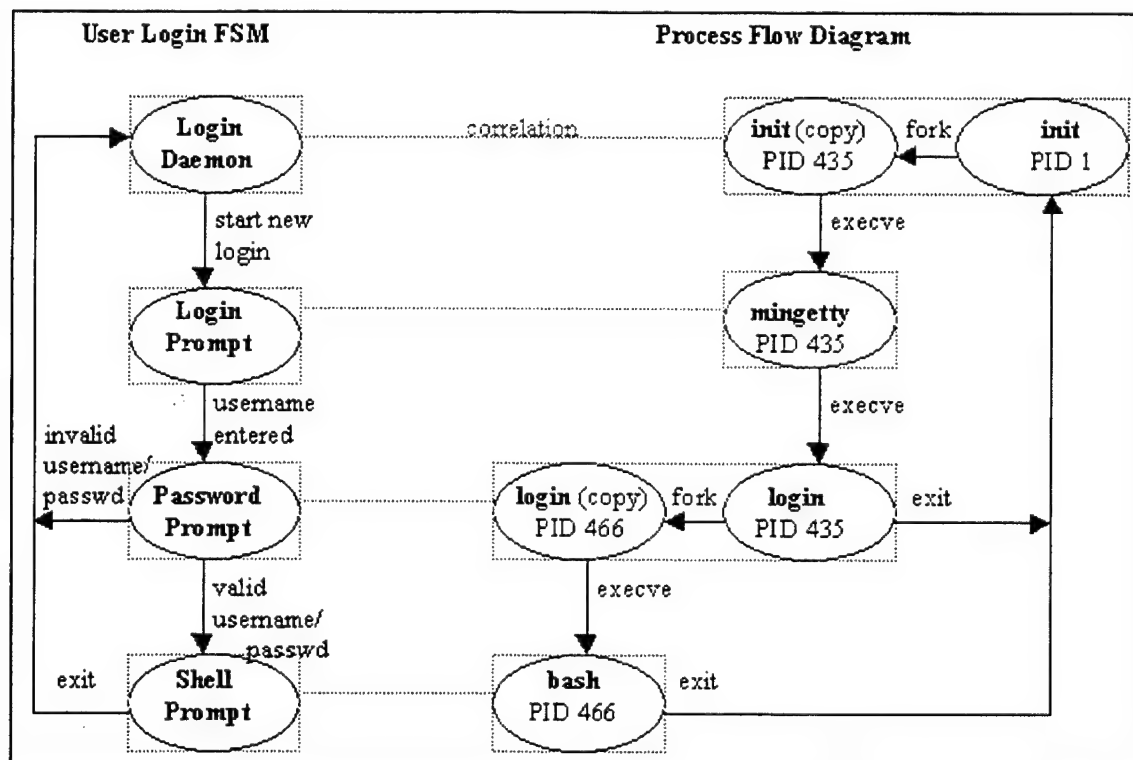


Figure 3. User Login Finite State Machine For Standard Linux

The 'init' process forks and adds the 'mingetty' application to the process table. The 'mingetty' application displays a login prompt to the user and waits for a username to be entered.

It is important to note that most Linux releases are defaulted to initialize more than one user terminal (tty) after startup. The 'init' process looks at a file called 'inittab' (normally located in the '/etc' directory) for a list of programs to run at startup [Ref.21]. Essentially, the 'inittab' file is the map that 'init' uses to set up the system and start at a specified run level. The 'inittab' file for Red Hat Linux 6.0 is configured to initialize 6 ttys. Each tty is set up by using the 'mingetty' application. The following example is for a system that is using only one tty. Figure 4 displays three important views to aid understanding how the user login process is started. First, an example of the process table is provided (the actual process table is much larger and includes all of the processes created by 'init' at startup, including the other 5 ttys). Second, an example of the tty display shows what the user can see at the terminal. Third, a snapshot of the finite state machine shows the location of the process execution flow at this point.

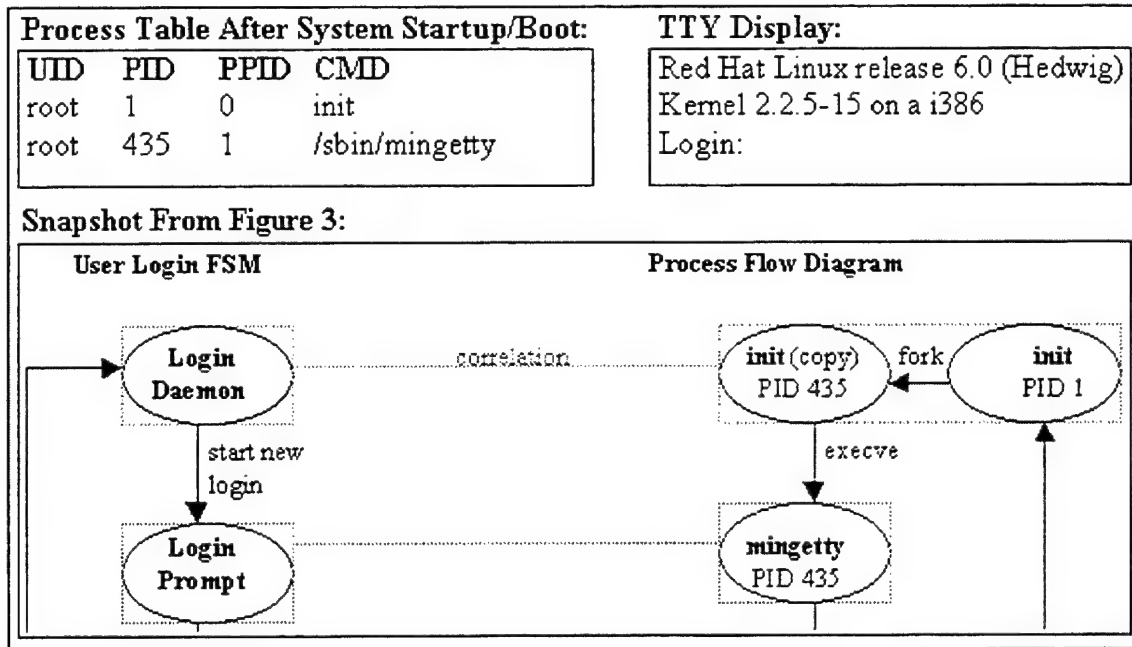


Figure 4. Process Table After System Startup/Boot

After a username is entered at the login prompt, 'mingetty' executes the 'login' application. The 'login' application displays a password prompt to the user and waits for a password to be entered. Figure 5 shows the process table, tty display, and process execution flow for this state. Note the PID is the same as it was for 'mingetty'. The 'login' application is not assigned a new PID because 'mingetty' used the 'execve' ("/bin/login") system call which allows another executable ('login') to take over the current position in the process table.

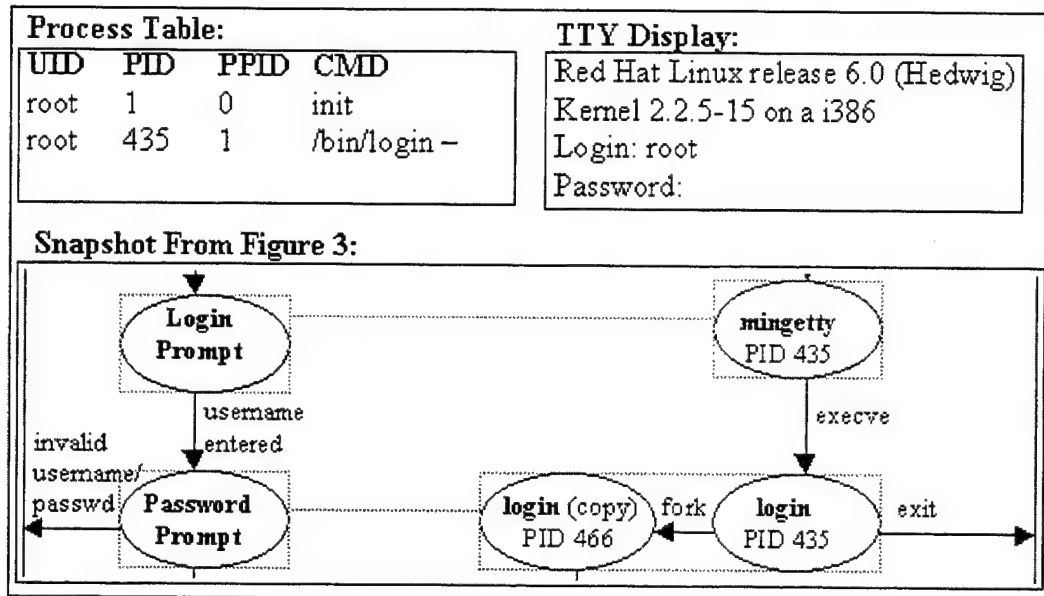


Figure 5. Process Table After Username Is Entered

If an invalid username/password pair is received, then 'login' exits and 'mingetty' is respawned by 'init'. When the 'login' application receives a valid username/password pair, then the shell application ('bash') is started as a separate process. Once the shell is started, then the user is considered logged into the system and is able to type commands at the terminal. Figure 6 shows the process table, tty display, and process execution flow for this state. Note that 'login' uses the 'fork' system call and creates a new PID before starting 'bash' with the 'execve' system call.

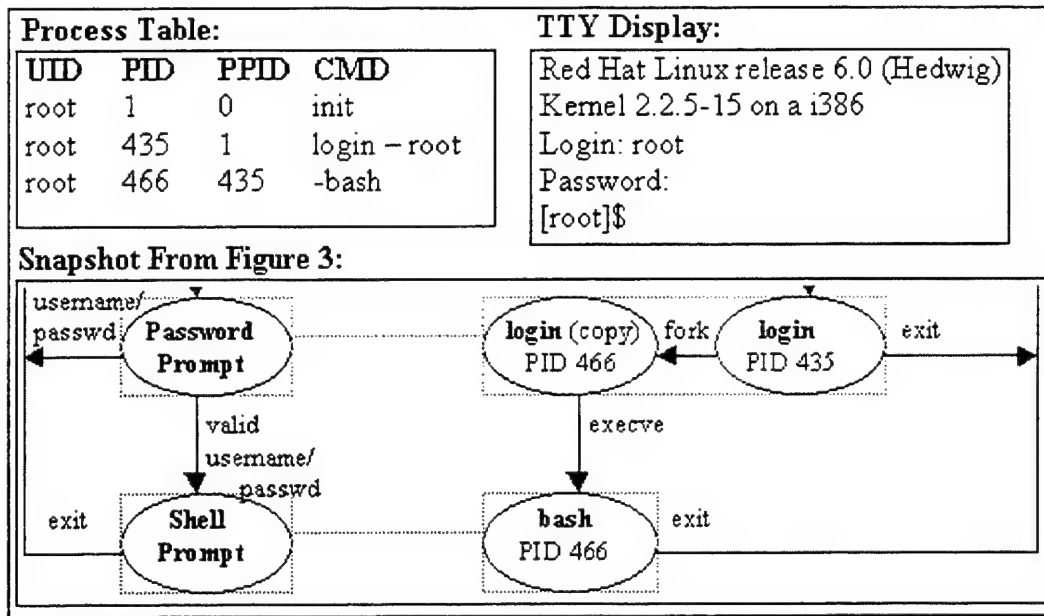


Figure 6. Process Table After Password Entered

The previous description of the Linux login process is a precursor to the next section that suggests modifications needed to implement a Trusted Path in Linux.

## 2. Modifications Needed

This section describes the modifications and additions required to implement the Trusted Path design from Chapter II. Figure 7 shows the process flow diagram for the Trusted Path.

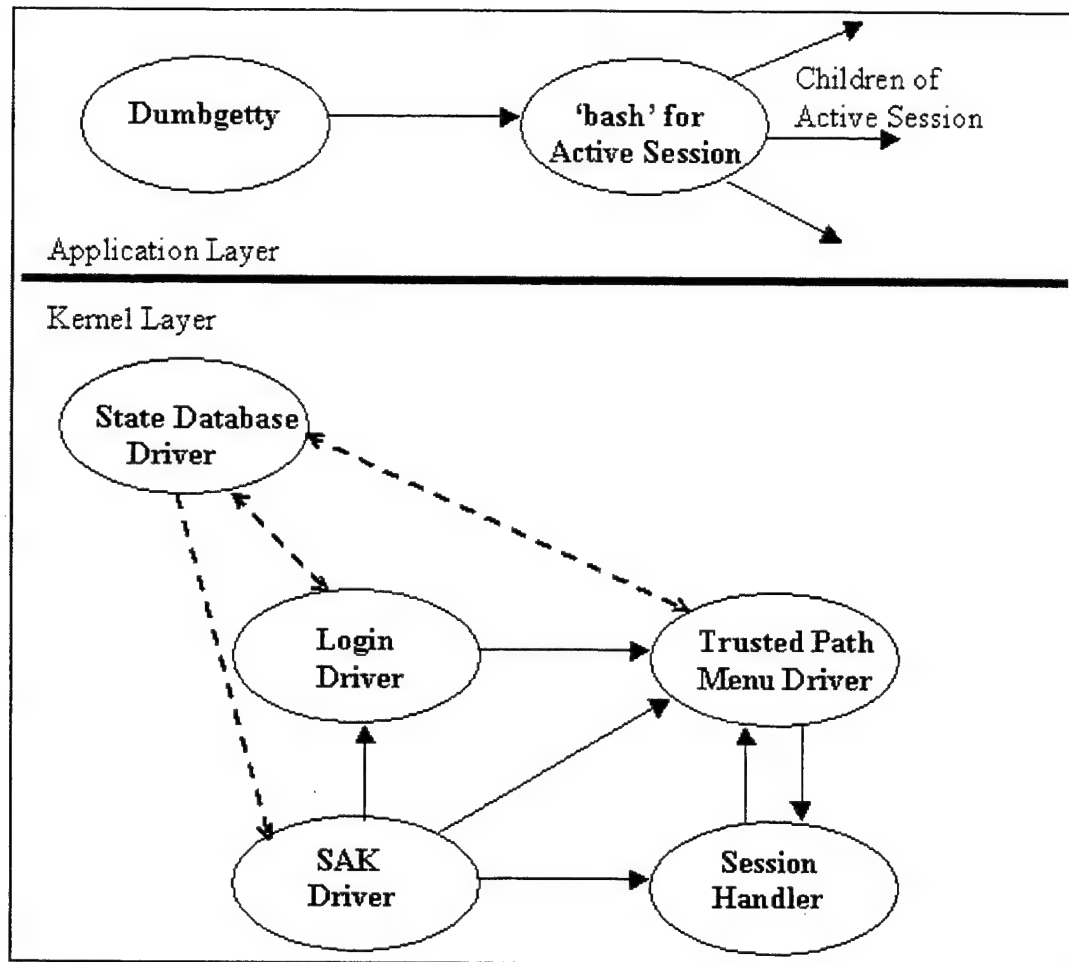


Figure 7. Trusted Path Process Flow Diagram

*a. Dumb Mingetty Process (dumbgetty)*

The dumb 'mingetty' or 'dumbgetty' process is a replacement for the current 'mingetty' process. The original 'mingetty' is being replaced because it is a user application and the Trusted Path must reside in the kernel. Therefore, 'mingetty' is no longer needed for login but something is needed to provide information to the user at system start up. The 'dumbgetty' program will provide a display that asks the user to press the SAK in order to invoke the Trusted Path User Interface. The login prompt can

be spoofed, so it is important to train users to always use the SAK to start a session. The 'dumbgetty' program will also serve (indirectly) as a parent process to newly created sessions. A parent process is required for all newly created processes. When a new session process family is created, the Session\_Driver (described later) will use the PID for 'dumbgetty' in order to masquerade as the parent process and create new children.

***b. Login\_Driver***

The source code for the Login\_Driver will be called 'login.c' and will reside in the '/linux/drivers/char/' directory. The code will be a combination of the original 'mingetty' and 'login' applications. The login module will represent and contain all of the functionality of the Login Prompt and Password Prompt states described in Chapter II. The original user applications such as 'mingetty' and 'login' can be easily modified and recompiled by a user, thus violating what little security is offered by Linux. In order to prevent a login spoof attack, this driver and the following suggested modules and drivers must reside in the kernel.

***c. State\_Module***

The source code for the State\_Module will be called 'state.c' and will reside in the '/linux/drivers/char/' directory. The code will contain the memory-resident variable that maintains the current state of the Trusted Path. All changes in state will be caused by one of the Trusted Path modules (Login or Trusted Path Menu). Therefore, it is necessary to implement the State\_Module in the kernel because it must be accessible to all of the Trusted Path modules.

To perform the correct action, the SAK\_Driver must know the state of the Trusted Path. The memory-resident variable contained in the State\_Module will be called 'TPState'. TPState will hold a number value based on the following possible states (described in detail in Chapter II):

- 1 - Start
- 2 - Ready
- 3 - Login Prompt
- 4 - Password Prompt
- 5 - Trusted Path Menu
- 6 - Help
- 7 - Session Level Prompt
- 8 - Change Password
- 9 - New Password
- 10 - Confirm and Change
- 11 - Run
- 12 - List Sessions
- 13 - Kill Session Menu

TPState will be referenced by the SAK\_Driver in order to learn the current Trusted Path state and move to the next correct state. The TPState variable will be updated after every state change.



*d. Session\_Database*

A database containing the active sessions and associated PIDs is necessary for keeping track of session information. The database will be used by the Session\_Driver (described later) to perform various session operations such as kill session, set session, and update current session. The database is a list of memory-resident structures containing session information for all of the available sessions. The structure is called “TPSession” and contains the following elements:

- active: is true when session is the active/current session
- PID: holds the PID of the first process in the session family
- label: holds the session level information, needed by Policy Enhanced Linux [Ref.19]

The Session\_Database will be encapsulated by the Session\_Driver described next.

*e. Session\_Driver*

The source code for the Session\_Driver will be called ‘session.c’ and will reside in the ‘/linux/drivers/char/’ directory. The Session\_Driver is an interface resident in the kernel to handle the creation, reactivation, suspension, and termination of user sessions.

The Session\_Driver encapsulates and manages the Session\_Database and is the only interface that will reference the Session\_Database. The declaration for the ‘TPSession’ structures (Session\_Database) is located in the Session\_Driver.

(1) Creation. The Session\_Driver looks in the Session\_Database to see if there is a session listed that matches the currentSession variable provided by the Trusted\_Path\_Menu\_Driver. If there is a matching session, then it must be reactivated (see next section on 'Reactivation'). If there is not a matching session, then a new session must be created. The new session is created based on the value of the currentSession variable. The Session\_Driver creates a new session by masquerading as 'dumbgetty', calling 'fork', and within the cloned child process calling 'execve' to start 'bash' for the requested session level. After the session is created a new TPSession structure is added to the Session\_Database and the 'active' flag is set. Also, if the 'active' flag was set for any other session, it is toggled to false. There can be only one active session.

(2) Reactivation. If the currentSession variable matches a session listed in the Session\_Database, then it must be reactivated. The Session\_Driver must alter some portion of the session process family structure to allow scheduling to resume. After the session is reactivated, the 'active' flag is set for the session in the Session\_Database. Also, if the 'active' flag was set for any other session, it is toggled to false at this time.

(3) Suspension. When the SAK is pressed and the previous state listed in the State\_Module is equal to Run (TPState = 10), then the Session\_Driver must suspend the active session process family from being scheduled.

(4) Termination. When the Kill Session Menu is used to kill a session, control is passed from the Trusted\_Path\_Menu\_Driver to the Session\_Driver and the process family for the selected session is deleted from the process table and the Session\_Database.

*f. Trusted\_Path\_Menu\_Driver*

The source code for the `Trusted_Path_Menu_Driver` will be called 'trustedmenu.c' and will reside in the '/linux/drivers/char/' directory. The code will represent and contain all of the functionality of the Trusted Path Menu and associated states (Help, Session Level Prompt, Change Password, List Sessions, Kill Session Menu) described in Chapter II. In addition, the `Trusted_Path_Menu_Driver` will have a memory-resident variable called 'currentSession' that keeps track of the current 'active' session.

*g. SAK Driver*

The SAK driver is described in the next section which provides a detailed description of how and where it be implemented.

**B. SECURE ATTENTION KEY (SAK)**

**1. Red Hat Linux 6.0 SAK**

There exists a function called 'do\_SAK' in the tty driver that, when called, abruptly kills all the tasks associated with the tty and forces mingetty to be respawned. The 'do\_SAK' function is found in the 'linux/drivers/char/tty\_io.c' source file [Ref.22]. The code comments listed prior to the 'do\_SAK' function state:

This implements the "Secure Attention Key" --- the idea is to prevent trojan horses by killing all processes associated with this tty when the user hits the "Secure Attention Key". Required for super-paranoid applications --- see the Orange Book for more details. [Ref.22]

The 'do\_SAK' function serves its purpose for those who know how and when to use it. However, it is not widely known by Linux users and is not a good example of how a

Trusted Path should function. The function 'do\_SAK' does not recognize the pressing of the "Secure Attention Key". The function 'do\_SAK' does implement what the SAK does after it has been pressed. There are at least two different ways of using the keyboard to invoke the 'do\_SAK' function and implement the SAK. These implementations are described in the following subsections.

*a. Magic System Request Key*

The 'Magic System Request Key' (MSRK) was implemented as a tool for Linux kernel programmers. The code for the MSRK is found in the 'linux/drivers/char/sysrq.c' source file [Ref.23]. The Red Hat Linux 6.0 default configuration for the Linux kernel has this option turned on. The MSRK can be turned on and off by reconfiguring and recompiling the kernel. The reconfiguration option is located in the 'Kernel Hacking' section of the Linux configuration menu [Ref.24]. The key sequence for the MSRK is 'Alt-SysRq' and one of the following keys:

- 'R' reset keyboard to XLATE mode
- 'K' call 'do\_SAK' using the current tty
- 'B' restart the machine immediately
- 'O' power off the machine
- 'S' emergency sync
- 'U' emergency remount
- 'P' show registers (pt\_regs)
- 'T' show task (process) information

'M' show memory information

'0'..'9' set console logging level

'E' terminate all user processes

'I' kill all user processes

'L' kill all processes including init

'Any other key' - prints the following help string:

```
Sync Unmount showPc showTasks showMem loglevel0-8 tErm kIll  
killalL
```

The call to the 'do\_SAK' function has been accomplished in the code for the 'K' option of the MSRK. The MSRK interface is called when the Alt-SysRq is pressed and then it waits for another keypress to perform the specific task. The Alt-SysRq sequence is recognized in the keyboard driver which is found in the 'linux/drivers/char/keyboard.c' source file [Ref.25].

The keyboard driver includes code that is only implemented if the MSRK has been turned on during kernel configuration. If the MSRK is enabled, the keyboard driver recognizes the Alt-SysRq key sequence (keycode = 84) and passes control to the MSRK before handing the keycode/scancode(s) to the current keyboard mode controller for processing [Ref.25]. The keyboard driver does the following:

- 1 - Receives scancodes created by key presses from the keyboard
- 2 - Converts the hexadecimal scancode(s) to a single decimal keycode
- 3 - Passes the scancode(s)/keycode to the current keyboard mode controller

- 4 - The keyboard mode controller passes the appropriate code(s) (ASCII, Unicode, keycode, raw scancodes) to the terminal driver [Ref.26].

The MSRK code in the keyboard driver bypasses steps 3 and 4 described above, effectively cutting off any chance for remapping the keycode to something else. Therefore, it is a good implementation of a SAK that communicates directly with the kernel. This will be the basis for the implementation of the SAK for Policy Enhanced Linux. Figure 8 shows how data flows through the keyboard driver with and without the MSRK enabled (the numbers (1), (2), (3), and (4) correspond to the steps described above).

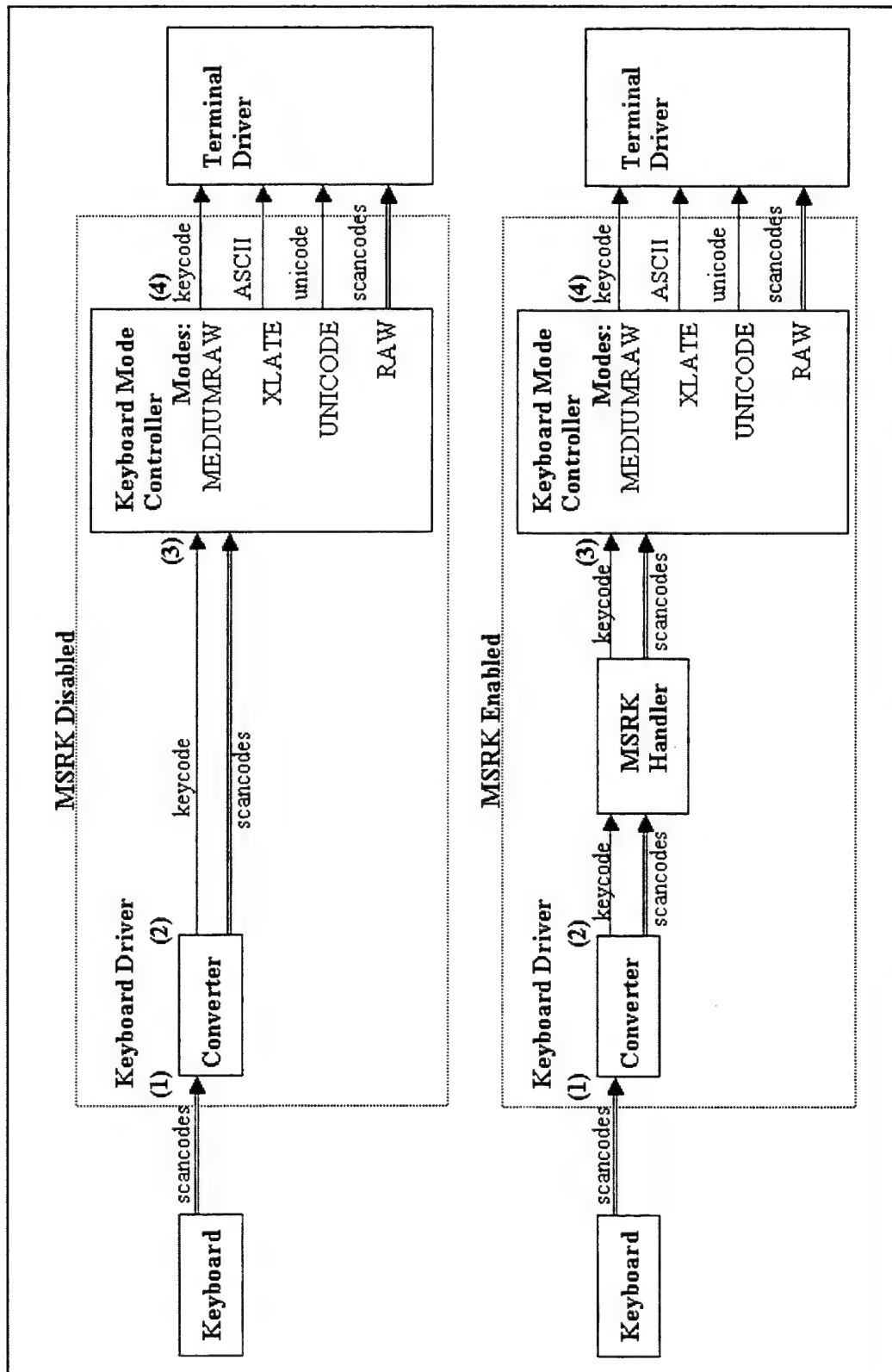


Figure 8. Keyboard Driver Data Flow

### *b. Loadkeys*

'Loadkeys' is a user application that was implemented for the main purpose of loading the kernel keymap when the keyboard is in translate (XLATE) mode [Ref.26]. The translate mode controller uses keymaps to assign keycodes to key symbols. Key symbols are evaluated and processed by the key handler in the keyboard driver. Then, the key handler passes the appropriate code(s) to the terminal (step 4 from the previous section) [Ref.26]. 'Loadkeys' provides a dynamic method of reassigning key symbols to keycodes. After 'loadkeys' is used, the keyboard driver's translation tables are updated.

There are two helper applications that are useful for creating a keymap to be loaded with 'loadkeys'. The 'dumpkeys' application is used to display the current contents of the keyboard driver's translation tables [Ref.28]. Specifically, 'dumpkeys -l' will print a list of all key symbols that can be used with 'loadkeys'. The 'showkey' application used with option 'k' ('showkey -k') will display the keycode(s) to the screen as the user presses the key(s) [Ref.29]. In summary, 'dumpkeys -l' will provide a list of key symbols, 'showkey -k' will provide the keycodes to specific keys, and 'loadkeys' can be used to link the keycodes and keysymbols together and update the keyboard driver's translation tables. Figure 9 shows how the keymap is used with 'loadkeys' to update the keyboard driver translation table. Figure 9 also shows how the helper applications 'showkey' and 'dumpkeys' are used to gather information for creating keymaps.



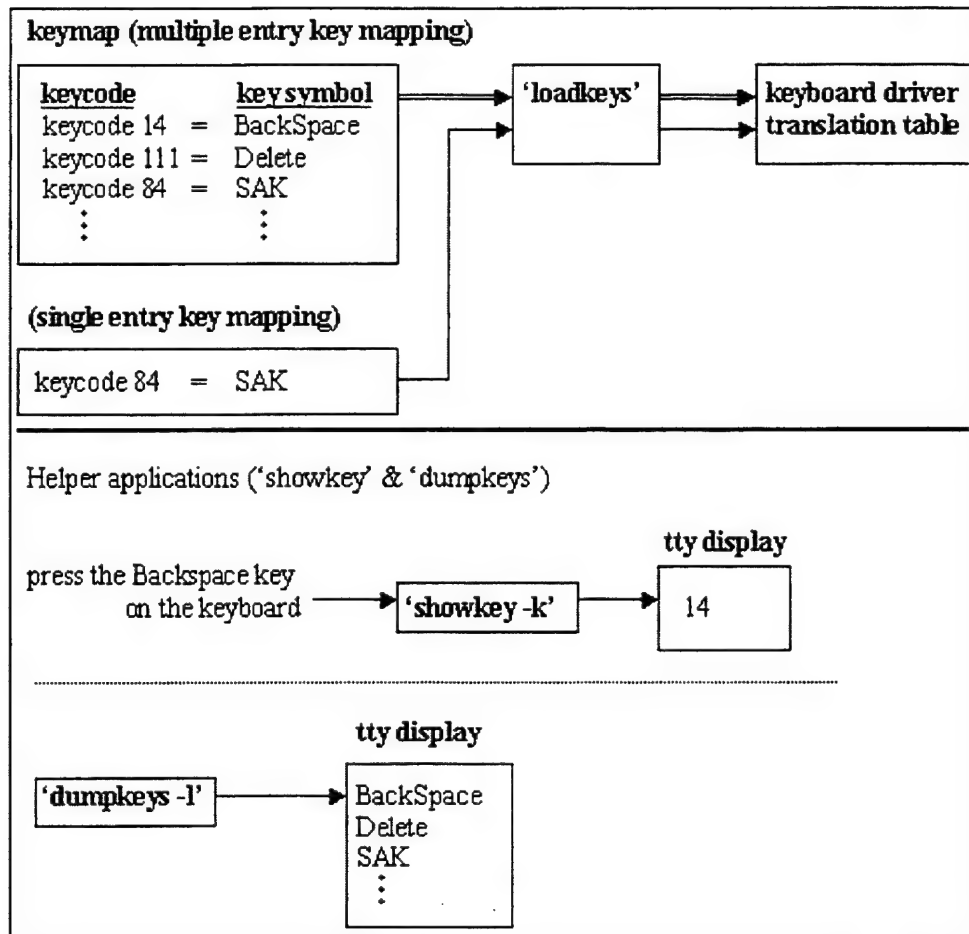


Figure 9. Loadkeys Description

'SAK' is a supported key symbol listed by 'dumpkeys -l' and when mapped to a keycode using 'loadkeys', performs the same as Alt-SysRq-K when the MSRk is enabled. It performs the same as the MSRk because the 'SAK' key symbol is defined in the special function table in the key handler. The SAK function in the key handler calls the 'do\_SAK' function in the terminal driver [Ref.22],[Ref.25]. Therefore, any keycode mapped to the 'SAK' key symbol using 'loadkeys' will implement the original Linux SAK.

The problem with 'loadkeys' is that it allows a user application to modify the keyboard driver's translation tables. A true SAK should not be modifiable by any means outside of the TCB. If the SAK were implemented using 'loadkeys', it would be easy to alter the SAK to perform a different action or launch a login spoof program. That is why the MSRK provides the best implementation for a SAK. Even if the original Linux SAK key symbol is assigned (using 'loadkeys') to the keycode for Alt-SysRq, the MSRK handler will catch the keycode and the keyboard mode controller will never see it unless the MSRK handler passes it on (which it does not do).

## **2. Modifications Needed**

### ***a. Magic System Request Key***

The Magic System Request Key provides a mechanism to invoke a kernelized SAK. The implementation of the MSRK will be used to provide a skeleton for implementing the SAK driver. The Trusted Path cannot have all of the destructive functionality inherent in the various options of the standard Linux MSRK. Therefore, only the 'K' option will be retained for implementation of the SAK. All other keys will default to 'do nothing'.

### ***b. SAK Driver***

The source code for the SAK driver will be called 'sak.c' and will reside in the '/linux/drivers/char/' directory. The 'K' option will be modified to call the appropriate interface (Login driver or Trusted\_Path\_Menu\_Driver) based on the current state provided by the State Database.

THIS PAGE INTENTIONALLY LEFT BLANK

## **IV. IMPLEMENTATION STATUS AND FUTURE WORK**

### **A. IMPLEMENTATION CONSIDERATIONS**

The SAK\_Driver and the dumbgetty process were implemented to show that the Secure Attention Key was possible. The SAK\_Driver was implemented using the Magic System Request Key (MSRK) driver as a skeleton. The MSRK and the SAK\_Driver rely on the keyboard driver ('linux/drivers/char/keyboard.c' & 'linux/drivers/char/pc\_keyb.c') to function appropriately. The MSRK uses the constant CONFIG\_MAGIC\_SYSRQ to enable and disable the appropriate code sections included in the keyboard driver. Therefore, a constant CONFIG\_SAK was created to enable and disable the SAK\_Driver in the keyboard driver. Everywhere in the keyboard driver that CONFIG\_MAGIC\_SYSRQ is defined, CONFIG\_SAK has also been defined. The definitions have been modified so that if both CONFIG\_MAGIC\_SYSRQ and CONFIG\_SAK are enabled, CONFIG\_MAGIC\_SYSRQ will be disabled. This ensures that the SAK\_Driver takes precedence over the MSRK and no resource conflicts will occur. However, it is still a good idea to disable the MSRK when the SAK\_Driver is enabled. The additions and modifications to the keyboard driver files are shown in Appendix B.

The 'make xconfig' command in X-windows was used to configure the kernel for recompilation during this implementation. The 'make xconfig' command launches the Linux kernel configuration menu. The configuration menu contains many sections (sub-menus) that list modules and drivers available for compiling when selected. One section,

Kernel Hacking, contains the options for enabling and disabling the MSR\_K. The script for the menu ('linux/scripts/kconfig.tk') was modified to include the Secure Attention Key (CONFIG\_SAK) in the Kernel Hacking section. Now the user has the ability to select the appropriate configuration option for enabling/disabling the Secure Attention Key during compilation. The modifications to the configuration menu script are shown in Appendix B.

The 'dumbgetty' process was implemented and is called by 'init' after all of the initialization processes have been started. As described in Chapter III, the 'inittab' file is used to tell 'init' which processes to run when initializing the system. The 'inittab' file was modified to call only 'dumbgetty' after initialization is complete. The original 'inittab' file was set up to start six separate ttys using the 'mingetty' application. The modified 'inittab' file is shown in Appendix B.

With the implementation of 'dumbgetty' and the SAK\_Driver, Linux now offers a reasonable user interface for the Secure Attention Key. When the system is started, 'dumbgetty' is called and the message "Press the Secure Attention Key (Alt-SysRq-K) to continue" is displayed to the screen. When the SAK is pressed, the SAK\_Driver handles the keycode and displays the message "SysRq: Congratulations, you've pressed the SAK". If the user presses any key other than 'K' in combination with 'Alt-SysRq', the message "Alt-SysRq-K is the SAK; press the SAK to continue" is displayed.

## **B. PROBLEMS ENCOUNTERED**

The major hurdle for this thesis was in determining where to put the Secure Attention Key. The research started with the 'Ctrl-Alt-Del' key sequence which is actually trapped by 'init' and assigned to the 'shutdown' application in the 'inittab' file. It was believed that by studying the 'Ctrl-Alt-Del' key sequence, some light may be shown as to how the SAK could be implemented. However, 'Ctrl-Alt-Del' turned out not to be the ideal place to look. The original Linux SAK is not a highly publicized item in the Linux documentation available on the Internet. Brouwer's publication "Kernel Korner: The Linux Keyboard Driver" was invaluable for understanding the inner workings of the keyboard [Ref.26]. Also extremely helpful was Brouwer's publication "The Linux keyboard and console HOWTO", written for the Linux Documentation Project [Ref.30]. After a considerable amount of time invested in reading the keyboard driver files and learning how keyboard input is handled in Linux, it was possible to move forward and design how the SAK would be implemented.

## **C. FUTURE RESEARCH**

### **1. Login\_Driver**

Final design and implementation decisions need to be made for the Login\_Driver. Research in this area could involve making a more efficient user interface for logging on to Linux at the kernel level (username and password encryption, password length/type restrictions, etc).

## **2. Session\_Driver**

Final design and implementation decisions need to be made for the Session\_Driver. Research in this area will involve a deep understanding of process family creation in Linux. Specifically, the ability of the Session\_Driver to masquerade as 'dumbgetty', call 'fork', and call 'execve' to start a new session.

## **3. Trusted\_Path\_Menu\_Driver**

Final design and implementation decisions need to be made for the Trusted\_Path\_Menu\_Driver. The Trusted\_Path\_Menu\_Driver references or is referenced by all of the other drivers in the Trusted Path. The design and implementation of this interface is not possible until the previous drivers have been completed.

## **4. Inittab**

The 'dumbgetty' file must be called by 'init' when the system has reached a ready state after booting. The 'inittab' file is vulnerable to attack by those who would seek to change the processes called by 'init'. Therefore, some research should be made into how 'dumbgetty' can be called by 'init' without using the 'inittab' script. Otherwise, it will be easy to circumvent 'dumbgetty' by modifying the 'inittab' file. The current way of preventing this flaw is by setting the permissions of 'inittab' to be only accessible by 'root'. In this way, only an administrator with 'root' permission would be able to modify 'inittab'.

## **5. Administrative Role**

The XTS-300 Trusted Path allows certain administrative options only to people with 'administrator' level privileges. Administrator functions such as auditing, managing

user space, creating compartments, etc. could be made available in the Trusted Path Menu when the user has logged in with a valid 'administrator' username and password.



THIS PAGE INTENTIONALLY LEFT BLANK

## APPENDIX A. MODULE DESIGN

### A. STATE\_MODULE

This module defines the externally visible structure of the State\_Module described in Chapter III. The module interface is used to read and modify the current state of the Linux Trusted Path. This module does not depend on any other module.

#### External Entry Points:

- SetTPState
- GetTPState

#### External Types and Constants:

#define START	1
#define READY	2
#define LOGIN_PROMPT	3
#define PASSWORD_PROMPT	4
#define TRUSTED_PATH_MENU	5
#define HELP	6
#define SESSION_LEVEL_PROMPT	7
#define CHANGE_PASSWORD	8
#define NEW_PASSWORD	9
#define CONFIRM_AND_CHANGE	10
#define RUN	11
#define LIST_SESSIONS	12

#define KILL\_SESSION\_MENU 13

typedef TPState\_type unsigned int

## 1. SetTPState

This entry point is used to set the current state of the Trusted Path.

### a. External Interface

int SetTPState(const TPState\_type desiredTPState);

### b. Inputs

- desiredTPState

The desired state to be set.

### c. Outputs

- <function result>

The success or failure of the operation. A value of NO\_ERROR indicates a success, while any other value indicates an error.

### d. Processing

Verify that the state to be set is valid by comparing the input desiredTPState with the defined constants. If the state to be set is valid, set the TPState variable to the desired state from the input desiredTPState. Return the value of NO\_ERROR as the function's return value. If the state to be set is not valid, return a value other than NO\_ERROR as the function's return value.

## **2.     GetTPState**

This entry point is used to read the current state of the Trusted Path.

### ***a.     External Interface***

TPState\_type GetTPState(void);

### ***b.     Inputs***

<none>

### ***c.     Outputs***

- <function result>

The current value of the TPState variable.

### ***d.     Processing***

Return the value of TPState as the function result.

THIS PAGE INTENTIONALLY LEFT BLANK

## APPENDIX B. SOURCE CODE

This appendix contains the source code for the SAK\_Driver that was implemented for Trusted Path Linux.

### A. SAK\_DRIVER

#### 1. Sak.h

The 'sak.h' file is located in the 'linux/include/linux/' directory.

```
// -----  
// File: sak.h  
// -----  
// Description: This is the header file for the SAK_Driver.  
// The skeleton for this code is provided by the Linux Magic  
// System Request Key Hacks 'sysrq.h' (c) 1997 Martin Mares  
// <mj@atrey.karlin.mff.cuni.cz> based on ideas by Pavel Machek  
// <pavel@atrey.karlin.mff.cuni.cz>.  
//  
// Created: 22-May-00 (S. Bartram)  
// Modifications:  
// -----  
  
#include <linux/config.h>  
  
struct pt_regs;  
struct kbd_struct;  
struct tty_struct;  
  
// Generic Alt-SysRq interface -- you may call it from any device  
// driver, supplying ASCII code of the key, pointer to registers  
// and kbd/tty structs (if they are available -- else NULL's).  
  
void handle_sysrq(int, struct pt_regs *, struct kbd_struct *,  
struct tty_struct *);
```

#### 2. Sak.c

The 'sak.c' file is located in the 'linux/drivers/char/' directory.

```
// -----  
// File: sak.c
```

```

// -----
// Description: This is the implementation file for the
// SAK_Driver.
// The skeleton for this code is provided by the Linux Magic
// System Request Key Hacks 'sysrq.c' (c) 1997 Martin Mares
// <mj@atrey.karlin.mff.cuni.cz> based on ideas by Pavel Machek
// <pavel@atrey.karlin.mff.cuni.cz>
//
// Created: 22-May-00 (S. Bartram)
// Modifications:
// -----

#include <linux/sak.h>
#include <linux/config.h>
#include <linux/sched.h>
#include <linux/interrupt.h>
#include <linux/mm.h>
#include <linux/fs.h>
#include <linux/mount.h>
#include <linux/kdev_t.h>
#include <linux/major.h>
#include <linux/reboot.h>
#include <linux/kbd_kern.h>
#include <linux/quotaops.h>
#include <linux/smp_lock.h>

#include <asm/ptrace.h>

#ifdef CONFIG_APM
#include <linux/apm_bios.h>
#endif

// -----
// Function:
//   handle_sysrq
// Inputs:
//   key          = 1 if a key has been pressed, = 0 if no key has
//                 been pressed
//   pt_regs      process table register structure
//   kbd          keyboard structure
//   tty          tty structure
// Outputs:
//   <none>
// Description: This function is called by the keyboard handler
// when Alt-SysRq is pressed. If no other keycode arrives,
// then the function returns and does nothing. If the keycode
// for "k" arrives, then the SAK has been pressed and a message
// is sent to the screen. If any other keycode arrives, the
// function returns and does nothing.
// -----
void handle_sysrq(int key, struct pt_regs *pt_regs,

```

```

        struct kbd_struct *kbd, struct tty_struct *tty)
{
    if (!key)
        return;
    printk(KERN_INFO "SysRq: ");
    switch (key) {
#ifdef CONFIG_VT
        case 'k':
            /* k -- SAK */
            printk("Congratulations, you've pressed the SAK\n");
            break;
#endif
        default:
            /* Unknown: help */
#ifdef CONFIG_VT
            if (tty)
                printk("Alt-SysRq-K is the SAK");
#endif
    }
}

```

## B. MODIFIED KEYBOARD DRIVER FILES

### 1. Keyboard.c

The 'keyboard.c' file is located in the 'linux/drivers/char/' directory.

#### a. Additions

The following additions were placed at the beginning of the file:

- // if both MSRQ and SAK are enabled, then disable the MSRQ
- #if defined(CONFIG\_MAGIC\_SYSRQ) && defined(CONFIG\_SAK)
- #undef CONFIG\_MAGIC\_SYSRQ
- #endif

The following addition encapsulates the #include <linux/sysrq.h>

statement:

- #ifdef CONFIG\_MAGIC\_SYSRQ
- #endif

The following addition is placed after the last addition:

- #ifdef CONFIG\_SAK



- #include <linux/sak.h>
- #endif

#### ***b. Modifications***

All instances of '#ifdef CONFIG\_MAGIC\_SYSRQ' were replaced with the following:

- #if defined(CONFIG\_MAGIC\_SYSRQ) ||  
defined(CONFIG\_SAK)

### **2. Pc\_keyb.c**

The 'pc\_keyb.c' file is located in the 'linux/drivers/char/' directory.

#### ***a. Modifications***

All instances of '#ifdef CONFIG\_MAGIC\_SYSRQ' were replaced with the following:

- #if defined(CONFIG\_MAGIC\_SYSRQ) ||  
defined(CONFIG\_SAK)

## **C. MODIFIED CONFIGURATION SCRIPT**

### **1. Kconfig.tk**

The 'kconfig.tk' file is located in the 'linux/scripts/' directory.

#### ***a. Additions***

All code dealing with the MSRK or containing the text CONFIG\_MAGIC\_SYSRQ was copied and modified accordingly.

Original text:

- write\_comment \$cfg \$autocfg "Kernel hacking"
- global CONFIG\_MAGIC\_SYSRQ

- write\_tristate \$cfg \$autocfg CONFIG\_MAGIC\_SYSRQ  
\$CONFIG\_MAGIC\_SYSRQ \$notmod

Addition immediately after original text above:

- global CONFIG\_SAK
- write\_tristate \$cfg \$autocfg CONFIG\_SAK \$CONFIG\_SAK  
\$notmod

Original text:

- set CONFIG\_MAGIC\_SYSRQ 0

Addition immediately after original text above:

- set CONFIG\_SAK 0

Original text:

- bool \$w.config.f 30 0 "Magic SysRq key"  
CONFIG\_MAGIC\_SYSRQ

Addition immediately after original text above:

- bool \$w.config.f 30 1 "Secure Attention Key"  
CONFIG\_SAK

## D. DUMBGETTY

### 1. Dumbgetty.c

The 'dumbgetty.c' file can be located anywhere. It has been compiled with the name of 'dumbgetty' and placed in the '/etc' directory for reference by the 'inittab' file.

```
// -----
// File: dumbgetty.c
// -----
// Description: This is the implementation file for the
// 'dumbgetty' process. 'dumbgetty' ignores all
// signals except SIG_KILL and SIG_STOP.
//
// Created: 1-Jun-00 (S. Bartram)
//
// Modifications:
// -----
```

```

#include <stdio.h>
#include <stdlib.h>

// -----
// Function:
//   main
// Inputs:
//   <none>
// Outputs:
//   <none>
// Description:
//   Workhorse function for 'dumbgetty'. Ignore all
//   signals that can be ignored. Display message to
//   screen and sleep until killed.
// -----
int main() {

    // ignore all of the signals that can be ignored
    signal(SIGHUP,SIG_IGN);
    signal(SIGINT,SIG_IGN);
    signal(SIGQUIT,SIG_IGN);
    signal(SIGILL,SIG_IGN);
    signal(SIGTRAP,SIG_IGN);
    signal(SIGIOT,SIG_IGN);
    signal(SIGBUS,SIG_IGN);
    signal(SIGFPE,SIG_IGN);
    signal(SIGUSR1,SIG_IGN);
    signal(SIGSEGV,SIG_IGN);
    signal(SIGUSR2,SIG_IGN);
    signal(SIGPIPE,SIG_IGN);
    signal(SIGALRM,SIG_IGN);
    signal(SIGTERM,SIG_IGN);
    signal(SIGCHLD,SIG_IGN);
    signal(SIGCONT,SIG_IGN);
    signal(SIGTSTP,SIG_IGN);
    signal(SIGTTIN,SIG_IGN);
    signal(SIGTTOU,SIG_IGN);
    signal(SIGURG,SIG_IGN);
    signal(SIGXCPU,SIG_IGN);
    signal(SIGXFSZ,SIG_IGN);
    signal(SIGVTALRM,SIG_IGN);
    signal(SIGPROF,SIG_IGN);
    signal(SIGWINCH,SIG_IGN);
    signal(SIGIO,SIG_IGN);
    signal(SIGPWR,SIG_IGN);

    printf( "Press the Secure Attention Key to continue:\n" );
    while (1) {
        sleep(1);
    }
}

```

```
    return 0;
}
```

## **E. MODIFIED INITTAB**

### **1. Inittab**

The 'inittab' file is located in the '/etc' directory.

#### ***a. Additions***

Original text:

- pr:12345:powerokwait:/sbin/shutdown -c "Power Restored; Shutdown Cancelled"

Addition immediately after original text above and just before the modifications in the next section:

- # Run 'dumbgetty'
- 1:2345:once:/etc/dumbgetty

#### ***b. Modifications***

Deleted out the following original text:

- # Run gettys in standard runlevels
- 1:2345:respawn:/sbin/mingetty tty1
- 1:2345:respawn:/sbin/mingetty tty2
- 1:2345:respawn:/sbin/mingetty tty3
- 1:2345:respawn:/sbin/mingetty tty4
- 1:2345:respawn:/sbin/mingetty tty5
- 1:2345:respawn:/sbin/mingetty tty6

THIS PAGE INTENTIONALLY LEFT BLANK

## LIST OF REFERENCES

1. Department of Defense, *Trusted Computer System Evaluation Criteria*, DOD 5200.28-STD, December 1985.
2. National Security Agency, *Computer Security Evaluation Frequently Asked Questions*, <http://www.radium.ncsc.mil/tpep/process/faq-sect1.html#Q8>, August 1999.
3. National Security Agency, *Computer Security Evaluation Frequently Asked Questions*, <http://www.radium.ncsc.mil/tpep/process/faq-sect1.html#Q2>, August 1999.
4. National Security Agency, *Computer Security Evaluation Frequently Asked Questions*, <http://www.radium.ncsc.mil/tpep/process/faq-sect1.html#Q3>, August 1999.
5. National Security Agency, *Common Criteria Version 2.1*, <http://www.radium.ncsc.mil/tpep/library/ccitse/ccitse.html>, March 1999.
6. National Security Agency, *Computer Security Evaluation Frequently Asked Questions*, <http://www.radium.ncsc.mil/tpep/process/faq-sect2.html#Q3>, August 1999.
7. National Security Agency, *Computer Security Evaluation Frequently Asked Questions*, <http://www.radium.ncsc.mil/tpep/process/faq-sect2.html#Q4>, August 1999.
8. National Security Agency, *Computer Security Evaluation Frequently Asked Questions*, <http://www.radium.ncsc.mil/tpep/process/faq-sect3.html#Q3>, August 1999.
9. Brinkley, Donald L., Schell, Roger R., *Concepts and Terminology for Computer Security*, ed. Abrams, Jajodia, and Podell, in *Information Security, An Integrated Collection of Essays*, IEEE Computer Society Press, 1995.
10. National Security Agency, *Trusted Product Evaluation Program*, <http://www.radium.ncsc.mil/tpep/epl/entries/TTAP-CSC-EPL-99-001.html>, November 1999.
11. National Security Agency, *Trusted Product Evaluation Program*, <http://www.radium.ncsc.mil/tpep/epl/entries/CSC-EPL-95-003.html>, July 1995.

12. Sun Microsystems, Inc., *Trusted Solaris Features, Evaluations*, [http://www.sun.com/software/solaris/trustedsolaris/ts\\_feature\\_eval.html](http://www.sun.com/software/solaris/trustedsolaris/ts_feature_eval.html), May 2000.
13. Sun Microsystems, Inc., *Trusted Facility Manual for Trusted Solaris 1.1, Vol.1*, Sun Microsystems Federal, Inc., February 1994.
14. Sun Microsystems, Inc., *Security Features User's Guide to Trusted Solaris 1.1*, Sun Microsystems Federal, Inc., February 1994.
15. National Security Agency, *Trusted Product Evaluation Program*, <http://www.radium.ncsc.mil/tpep/epl/entries/CSC-EPL-92-003-D.html>, March 1998.
16. National Security Agency, *Trusted Product Evaluation Program*, <http://www.radium.ncsc.mil/tpep/epl/historical.html>, May 1992.
17. Wang Federal, Inc., *XTS-300 User's Manual, STOP 4.4 Version*, Wang Federal, Inc., April 1997.
18. Wang Federal, Inc., *XTS-300 Trusted Facility Manual, STOP 4.4 Version*, Wang Federal, Inc., April 1997.
19. Clark, Paul C., *A Linux-Based Approach to Low-Cost Support of Access Control Policies*, United States Navy, Naval Postgraduate School, September 1999.
20. Beck, Michael, Bohme, Harald, Dziadzka, Mirko, Kunitz, Ulrich, Magnus, Robert, Verworner, Dirk, *Linux Kernel Internals, Second Edition*, Addison-Wesley, 1998.
21. Wirzenius, Lars, Oja, Joanna, *The Linux System Administrator's Guide, Version 0.6.2*, <http://www.linuxdoc.org/LDP/sag/x1766.html>, October 1999.
22. Torvalds, Linus, *Red Hat Linux 6.0*, linux/drivers/char/tty\_io.c, 1992.
23. Mares, Martin, *Red Hat Linux 6.0*, linux/drivers/char/sysrq.c, 1997.
24. Torvalds, Linus, *Red Hat Linux 6.0*, xconfig, 1999.
25. Torvalds, Linus, Myreen, Johan, Niemann, Christoph, Kankkunen, Risto, Brouwer, Andries, Mares, Martin, Uytterhoeven, Geert, *Red Hat Linux 6.0*, linux/drivers/char/keyboard.c, 1998.
26. Brouwer, Andries E., *Kernel Korner: The Linux Keyboard Driver*, Linux Journal Issue #14, June 1995.

27. Torvalds, Linus, *Red Hat Linux 6.0*, loadkeys, October 1997.
28. Torvalds, Linus, *Red Hat Linux 6.0*, dumpkeys, October 1997.
29. Torvalds, Linus, *Red Hat Linux 6.0*, showkey, October 1997.
30. Brouwer, Andreas E., *The Linux keyboard and console HOWTO*,  
<http://www.redhat.com/mirrors/LDP/HOWTO/Keyboard-and-Console-HOWTO.html>, February 1998.



THIS PAGE INTENTIONALLY LEFT BLANK

## INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..... 2  
8725 John J. Kingman Rd., STE 0944  
Ft. Belvoir, Virginia 22060-6218
  
2. Dudley Knox Library..... 2  
Naval Postgraduate School  
411 Dyer Rd.  
Monterey, California 93943-5101
  
3. Chairman, Code CS..... 1  
Computer Science Department  
Naval Postgraduate School  
Monterey, California 93943-5000
  
4. Dr. Cynthia E. Irvine ..... 3  
Computer Science Department Code CS/Ic  
Naval Postgraduate School  
Monterey, CA 93943-5000
  
5. Mr. Paul C. Clark ..... 3  
Computer Science Department Code CS/Cp  
Naval Postgraduate School  
Monterey, CA 93943-5000
  
6. Mr. James P. Anderson ..... 1  
James P. Anderson Company  
Box 42  
Fort Washington, Pennsylvania 19034
  
7. Mr. Paul Pitelli ..... 1  
National Security Agency  
Research and Development Building  
R2, Technical Director  
9800 Savage Road  
Fort Meade, Maryland 20755-6000

8. Mr. Howard Holm ..... 1  
National Security Agency  
Research and Development Building  
R23, Chief  
9800 Savage Road  
Fort Meade, Maryland 20755-6000
9. Mr. Grant Wagner ..... 1  
National Security Agency  
Research and Development Building  
R23  
9800 Savage Road  
Fort Meade, Maryland 20755-6000
10. Carl R. Siel ..... 1  
Space and Naval Warfare Systems Command  
PMW 161  
Building OT-1, Room 1024  
4301 Pacific Highway  
San Diego, California 92110-3127
11. Commander, Naval Security Group Command..... 1  
Naval Security Group Headquarters  
9800 Savage Road  
Suite 6585  
Fort Meade, Maryland 20755-6585
12. Ms. Barbara Fleming..... 1  
Defense Information Systems Agency  
Columbia Pike, Suite 400  
Falls Church, Virginia 22041-3230
13. Mr. Richard Hale..... 1  
Defense Information Systems Agency  
Columbia Pike, Suite 400  
Falls Church, Virginia 22041-3230
14. Col. Timothy Fong ..... 1  
Defense Information Systems Agency  
5600 Columbia Pike  
Falls Church, Virginia 22041

15. Mr. George Bieber..... 1  
Defense Information Systems Agency  
Center for Information Systems Security  
5113 Leesburg Pike, Suite 400  
Falls Church, Virginia 22041-3230
16. Ms. Louise Davidson..... 1  
N643  
Presidential Tower 1  
2511 South Jefferson Davis Highway  
Arlington, Virginia 22202
17. CAPT James Newman ..... 1  
N64  
Presidential Tower 1  
2511 South Jefferson Davis Highway  
Arlington, Virginia 22202
18. ENS Scott A. Bartram ..... 3  
C/O Barbara Sullivan  
P.O. Box 6  
Crescent Lake, OR 97425